Collaborative Filtering Movie Recommender System

Hongbin Qu
University of Washington
Seattle, WA
hongbinqu9@gmail.com

Abstract

Recommender systems play an important role in the information era. It can be applied in various aspects such as music apps, movie websites, social media, etc. Collaborative filtering is a common approach to design a recommender system. This paper discusses several ways to implement memory-based collaborative filtering on Movielens dataset and evaluates their performance.

1 Introduction

There are generally three recommendation types: collaborative filtering, content-based filtering, and hybrid recommender system (the combination of the first two approaches).[1] The core idea of collaborative filtering algorithms is the similarity of two users' past preference can predict a user's future preference. In this project, couples of collaborative filtering algorithm are performed. The data will be analyzed is MovieLens datasets[2], which is a rating dataset contains 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users.

2 Collaborative filtering implementation

2.1 Memory-based CF

2.1.1 Similarity matrix method

In order to recommend movies to a certain user, the preference of the user should be explored and the user's ratings on different movies would be predicted accordingly. Then movies with the highest predicted rating would be recommended to the user. Firstly, the user-item rating matrix R is built, with a size of $m \times n$, where m is the number of users and n is the number of movies with at least one rating by users. It can be calculated that the sparsity of the rating matrix R is 1.7%. In a user-based approach, a user similarity matrix, containing the similarity of the relationship between every two individuals, is built up in the beginning. Afterward, makes a prediction based on the matching result of the rating matrix. Cosine distance, calculated as the dot product between two vectors divided by their magnitudes, is used as the metric of similarity between two users:

$$sim(x, y) = \frac{x \cdot y}{||x|| \times ||y||} = \sum_i \frac{x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

Therefore, movie vectors pointing in a similar direction will get a higher similarity score. Then the $m \times m$ similarity matrix S represents the similarity matrix.

Splits the dataset as training set and testing set with a ratio of 9:1 and trained the data on the training set. In order to alleviate the impact of user's biased rating, the rating matrix should be demeaned. Assume $R_{i,j}$ is the value of rating that user i provided for movie j. The prediction of user i's rating

on movie j, $P_{i,j}$, can be denoted by the sum of user i's similarity with all other users multiplied by all other users' rating on movie j.[3]

$$P_{i,j} = \overline{R}_i + \frac{\sum_{k=i}^{n} S_{i,k}(R_{k,j} - \overline{R}_k)}{\sum_{k=i}^{n} S_{i,k}}$$

After the prediction matrix P is generated, the performance of this algorithm can be evaluated by calculating the root mean square error between the specified ratings in R and the corresponding predicted ratings in P. RMSE of the training set is 3.18 and the RMSE of the testing set is 3.65.

For comparison, the item-based collaborative filtering algorithm is also implemented. The implementation of item-based CF follows a similar procedure as stated above. The similarity matrix S now represents the similarity between every two movies. The prediction of user i's rating on movie j is given as movie j's similarity with all other movies multiplied by user i's rating on all movies.

$$P_{i,j} = \frac{\sum_{k=i}^{n} R_{i,k} S_{k,j}}{\sum_{k=i}^{n} S_{k,j}}$$

After applying the algorithm, the performance can also be evaluated by RMSE, which is 3.43 for the training set and 3.64 for the testing set.

2.1.2 K Nearest Neighbor method

In this section, a user-based collaborative filtering based on kNN algorithm is implemented. The main task of kNN CF algorithm is to find the k nearest neighbors for user i. The more similar two users' preference is, the more similar the two users would be. Also, cosine distance metric is used to define the similarity between rating vectors. 10-fold cross validation is also applied to this analysis, and the range of k starts from 2 to 50 with a step size of 2. With the facilitate of "surprise", a Python scikit building and analyzing recommender systems,[4] the testing RMSE against k and the ROC curve are plotted, which are shown in Figure 1 and Figure 2.

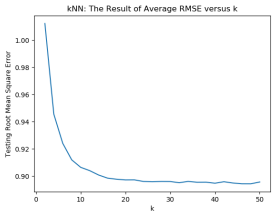

Figure 1: kNN with cosine distance

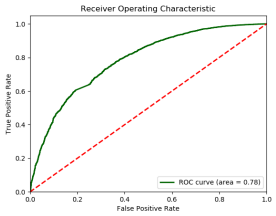

Figure 2: kNN ROC with a threshold of 3

2.2 Model-based CF

Model-based collaborative filtering is establishing models to predict users' rating of unrated movies. In this case, latent factor models are used to reduce the dimensionality of the model by compressing user-movie matrix into a low-dimensional representation in terms of latent factors.

2.2.1 Non-negative Matrix Factorization

A non-negative matrix factorization method is implemented to complete the rating matrix. The core idea is to factorize the rating matrix R into two low-dimensional matrices U and V based on the number of latent features, with all elements of the matrices are non-negative.[5] U matrix and V matrix map users and movies into a set of hidden factors respectively. The optimization objective can be formulated as:

$$min_{U,V} \sum_{i=1}^{m} \sum_{j=1}^{n} (R_{i,j} - (UV^T)_{i,j})^2$$

subjects to $U \geq 0$ and $V \geq 0$, which can be regraded as a convex optimization problem. Setting the number of latent factors to sweep from 2 to 50 with a step size of 2. The curve showing the average

RMSE via 10-folds cross validation is plotted in Figure 4. As is shown, the optimal number of latent factors is 16, which minimizes the error. In the optimal case, the ROC curve with a threshold of 3 is also plotted in Figure 5, whose AUC is 0.78.

For the 16 latent factors, it can be interpreted as 16 different characteristics about each movie. The genres of top 10 movies of the first latent vector is extracted, which are "Action|Crime|Horror|Sci-Fi|Thriller", " Drama", "Drama|Horror|Mystery|Thriller", "Adventure|Animation|Comedy", "Comedy|Thriller", "Horror|Thriller", "Comedy", "Drama", "Documentary", "Comedy|Drama". It is not explicitly known what this factor represents, but it can be inferred that it might have something to do with comedy or thriller movies, because they have a relatively high frequency.

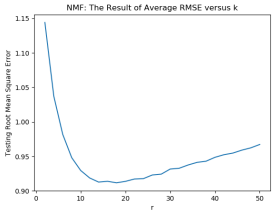

Figure 3: NMF with cosine distance

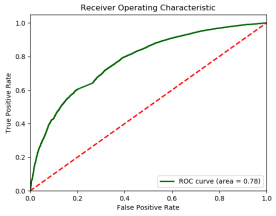

Figure 4: NMF ROC with a threshold of 3

2.3 Neural Network

In this section, a deep learning method is applied to build up a recommendation system. A feedforward neural network was built up, with Shioulin's essay[6] and Rosenthal [7] as a reference, which is also based on matrix factorization as other methods applied in this report. The neural network structure is shown in Figure 5.

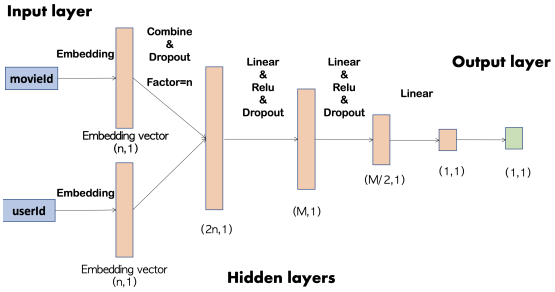

Figure 5: Neural Network Structure

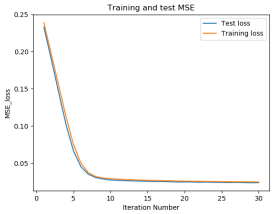

Figure 6: Training and Test MSELoss

Input layer: The input of the neural network is a pair or pairs (depends on batch size) of userId-movieId. And the model will generate users and movies embedding, so it needs the number of unique userId, movieId and the rating value of each pair, in this case, a rating matrix as above needs to be generated before training, so all the missing values are replaced by zero.
Hidden layers: The first hidden layer would generate users and movies embedding to map them with several factors. The number of factors is set to be a parameter, so that we can adjust it to find how it influences recommendation quality. The input of the first hidden layer are two scalars and the output would be two embedding vectors with size of $n \times 1$.
The second layer then would combine the two embedding vectors and use dropout method to reduce overfitting. So the output of this layer would be a vector with a size of $2n \times 1$.
The third, forth and fifth layers are all linear network layers. In the third and forth layer, Relu and dropout functions are also applied to introduce non-linearity and reduce overfitting.
Output layer: The output of the whole neural network would be a single scalar for each pair of userId-movieId, which is the prediction result.
Loss function and optimizer: The loss function is MSELoss. The optimizer is SGD.
Hyperparameters: There are two hyperparameters in optimizer, they are learning rate and

momentum. We provided a range for learning rate and momentum, and in training processing, they are randomly picked. $lr = 10^a, a \in \{-7, -6 \ldots -1\}$ and momentum$\in (0.4, 0.99)$. Also, there are two additional parameters, the first one is the number of factors, N, in the first hidden layer, and the second one is the output size of the third hidden layer, M. The output size of the third layer is automatically set as M/2.

Process and Results: The training was processed with Pytorch, with a batch size of 50. The training process was performed in 20 trials, and there are 30 iterations in each trial. Before each trial, the network would randomly generate hyperparameters and be refreshed. Figure 5 shows the plot of MSELoss versus Iteration number in one of the trials. In this trial, $lr = 1e-5$, $momentum = 0.4$, $M = 42$, and number of embedding factor is 50. The minimum training loss is 0.023 and the minimum test loss is 0.024. The test error is 1.14.

3 Result and Analysis

Comparing the results of the memory-based similarity matrix approach, they both have very large errors. It is not surprising because the rating matrix is extremely sparse, most of the entries have no rating, making the prediction inaccurate. Therefore, this approach is quite intuitive but have very bad performance.

To get rid of this problem, the k nearest neighbor algorithm is used to take only the k most similar vectors into account. In fact, the similarity matrix approach is the extreme situation of the kNN method when k is the number of all the user vector. By choosing an appropriate value of k, the errors can be lowered to less than 1. In addition, the ROC curve is used to evaluate the performance of the filter. It's known that the bigger the area under the curve (AUC) is, the more accurate a binary classifier is. In this case, a threshold is set to divide the rating scores into two categories, converting the continuous system into a binary system. It is shown that the AUC is 0.78. KNN approach does not require prior knowledge about the dataset has, and makes a relatively good prediction but when it comes to a big and sparse dataset, this method will be extraordinarily inefficient and memory-demanding. Therefore, s matrix factorization approach should be implemented.

By applying non-negative matrix factorization, latent features can be extracted from the dataset. Although the accuracy does not improve a lot compared to the KNN approach, NMF is still useful to deal with a very sparse matrix with many ambiguous attributes, such as the genres of movies in this case.

According to the loss-iteration plot in Figure 6, it can be found that training loss and test loss had an extreme decrease in the first 7 iterations, and then the tendency suddenly became gentle. From the result of deep learning method, it can be found that the training loss and testing loss both are numerically low, but the error is large, which doesn't meet the expectation. But the model is based on matrix factorization, which means that a large amount of missing ratings are replaced by zero, meanwhile, after the first iteration of training, it can be found that almost all values that the model predicted were close zero, due to the initial variables setup. Those missing values replaced by zero also misleads the model, so it believes that the error is small and the training performance is wonderful, but the true error is still big. In order to improve the actual predicting accuracy, a more reasonable network only based on existing data should be constructed.

4 Conclusion

To summarize, the root mean square error of each approach is shown in the table below:

Approach	user-based cf	item-based cf	kNN	MFN	NN
RMSE	3.65	3.64	0.88	0.91	1.08

References

[1] Patel, Khagesh, and Ankush Sachdeva. "Hybrid Recommendation System." (2014).

[2] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

[3] Schafer, J. Ben, et al. "Collaborative filtering recommender systems." The adaptive web. Springer, Berlin, Heidelberg, 2007. 291-324.

[4] Welcome to Surprise' documentation!. (n.d.). Retrieved from https://surprise.readthedocs.io/en/stable/index.html

[5] Lee, Daniel D., and H. Sebastian Seung. "Learning the parts of objects by non-negative matrix factorization." Nature 401.6755 (1999): 788.

[6] Shioulin."PyTorch for Recommenders 101", 10 April 2018, https://blog.fastforwardlabs.com/2018/04/10/pytorch-for-recommenders-101.html

[7] EthanRosenthal."torchmf", 18 May 2017, https://github.com/EthanRosenthal/torchmf/blob/master/torchmf.py

Appendices

All other graphs, tables, sources and code lists are in:
https://github.com/robin-qu/Movie-Recommender